УДК 681.3 http://doi.org/10.15407/pp2024.02-03.037

Д.В. Рагозін, А.Ю. Дорошенко

МОДЕЛЮВАННЯ ПРОДУКТИВНОСТІ ПАМ'ЯТІ ВІДЕОКАРТ ДЛЯ LLM-НЕЙРОМЕРЕЖ

У статті розглянуто характеристики швидкодії нейромережевих алгоритмів типу Generative pre-trained transformer, відомих також як Large Language Model, на сучасній відеокарті з метою визначення обмежень на використання таких нейромереж на мобільних обчислювальних платформах. Нейромережі цього класу цікаві як засіб ухвалення рішень, але на великому корпусі вивчених текстів їхня швидкодія уповільнюється і швидко зростає кількість необхідних обчислюваних ресурсів, тому корисно визначити, чи можливе використання таких нейромереж, сформованих на звуженому корпусі текстів і на пристроях з відносно невеликою обчислювальною потужністю. Дослідження характеристик здійснювалося за допомогою симулятора GPGPUSim, який може бути вільно сконфігурований як віртуальна відеокарта будь-якої потужності. Оскільки алгоритм базований на послідовності множення матриць, а його швидкодія на сучасних обчислювачах обмежена швидкодією підсистеми пам'яті, була досліджена статистика роботи кеш-пам'яті відеопроцесора різних рівнів та її взаємодія з основною пам'яттю системи. За допомогою GPGPUSim зібрано статистику для різних варіантів конфігурації мережі Generative pre-trained transformer версії 2, від конфігурації small до конфігурації xl. Аналіз статистики доступів до кеш-па'мяті другого рівня, промахів при доступі до кеш-пам'яті, а також кількості доступів у основну динамічну пам'ять показує, що у середніх конфігураціях кількість промахів у кеш другого рівня перевищує 7-8%, що досить багато. Це свідчить, що наявного об'єму кеш-пам'яті другого рівня не вистачає для функціонування основних алгоритмів і наявний досить значний обмін з основною пам'яттю. Натомість мінімальна конфігурація small може бути обрахована з мінімальним залученням ресурсів, тому може бути використана у перспективі як система прийняття рішень на платформах з невеликими ресурсами у разі зменшення корпусу текстів. А це відкриває цікаві перспективи для використання нейромереж цього типу для автономного ухвалення рішень.

Ключові слова: відеокарта, large language model, швидкодія, нейромережа, обчислювальні ресурси

D.V.Rahozin, A.Y. Doroshenko

MODELING VIDEOCARD MEMORY PERFORMANCE FOR LLM NEURAL NETWORKS

The paper covers the analysis of performance characteristics of neural network-based algorithms class Generative pre-trained transformer, also known as Large Language Model, for contemporary videocards. The goal is to check the application limitations for this class of neural networks for mobile computing platforms. This network class is interesting for use as control system tool, but for the bigger text corpuses the network performance degrades and the number of used computer resources grows quickly, so we need to explore if this type of network, but based on a smaller text corpus, is a feasible tool for devices with comparatively low computing capability. The performance investigation was performed with the help of GPGPUSim simulator, which can be freely configured as a virtual videocard of any computing capability. As this neural network computations are based on the calculation of a sequence of matrix multiplications, and its performance is limited by the memory bandwidth, we analyze the behavior statistics of the different cache memory levels of the videocard processor and the cache interaction with the main memory. With the help of GPGPUSim we have gathered statistics for different Generative pre-trained transformer version 2 configurations, from small to xl configurations. The level 2 cache memory access statistics, level 2 cache memory access misses, number of accesses to main memory show that even for the middle-level network configurations the number of level 2 cache memory misses exceeds the 7-8% level. This number looks high and this evidence shows that the size of the cache memory is quite small for executing this neural network configuration, also there is the substantial traffic from cache memory to main memory. Although, the minimal so-called small configuration can be computed faster and with moderated resources, and so can be used further as a part of decision-making system for computing platforms with moderate performance and resources for the case of limited text corpuses. This opens good enough perspectives for using this type of neural networks for autonomous decision making.

Key words: videocard, large language model, computing performance, neural network, computational capability

Вступ

В останні 10 років розвиток мікроелектроніки практично привів до виникнення комп'ютерної інженерної галузі, наближеної до поняття «штучний інтелект». Якщо 7-8 років тому складні завдання розпізнавання образів або сегментації зображення у реальному часі вважалися за фантастику навіть для спеціалізованих потужних відеопроцесорів, то зараз навіть середні, а не флагманські мобільні обчислювальні платформи, такі як, System-on-Chip (SoC) від Qualcomm серії Snapdragon, вирішують ці завдання в реальному часі. Втім, новітні нейромережі типу LLM[1] кидають виклик наявній потужності обчислювачів. Розвиток мікроелектроніки за законом Мура прогнозує експоненціальне збільшення кількісті транзисторів у рамках мікросхем SoC і впровадження ще більш потужних нейромереж на мобільній обчислювальні платформи у найближчі роки. Однак, за прогнозами складні LLM буде застосовуватися обмежено.

У цій статті ми досліджуємо LLM-нейромережі з точки зору їх застосовності на мобільних платформах. Ми дослідимо найцікавіші характеристики нейромережі – швидкодію доступу до пам'яті і ефективність роботи багаторівневої пам'яті відеокарти для наявних алгоритмів. Більша кількість транзисторів принципово дозволяє мати додаткові обчислювальні пристрої, але їхня робота неможлива без ефективного доступу до пам'яті системи. Потенційний масивний обчислювальний паралелізм відеокарти може нівелюватися складнощами доступу до пам'яті, не зважаючи на наявні апаратні пристрої для прискорення доступу до пам'яті або приховування затримок під час доступу до пам'яті. Аналіз за допомогою спеціалізованих симуляторів відеокарт дозволяє оцінити використання ієрархії пам'яті та можливості застосування алгоритму на певних конфігураціях відеокарт.

Обчислення сучасних нейромереж

Практично всі сучасні нейромережі базуються на операціях множення матриць [2], що дозволяє успішно формалізувати обчислювальні процеси, коли нейромережа подається як послідовність обчислювальних блоків, які обробляють матриці. Якщо колись множення матриць було обчислювально складною операцією, оскільки потребувало арифметичного блоку з обробкою плаваючої коми, то зі збільшенням кількості транзисторів конвеєризоване множення чисел із плаваючою комою обчислюється за 1 машинний цикл навіть у дешевих процесорах ARM. З іншого боку, кожне множення потребує доступу до двох комірок пам'яті з умовних матриць A та B, що вже проблематично виконати за 1 цикл, за винятком архітектур DSP, які зараз застосовуються нечасто і лише як спеціалізовані співпроцесори.

Таким чином, ефективність подібного сорту алгоритмів повністю залежить від підсистеми пам'яті: від ефективності алгоритму, який має заощаджувати доступи до пам'яті, та від правильного керування доступом до ієрархічної пам'яті відеокарти. У більшості випадків програмування алгоритмів такі оптимізації не проводяться.

Питання, пов'язані з керуванням пам'яттю відеокарти для старіших нейромереж типу YolovX описані в [3]. У наш час за два роки може змінюватися покоління відеокарт, тому подібні за структурою нейромережі вже давно виконують свої функції у реальному часі, і фокус зацікавлення зміщений на більш обчислювально важкі системи, зокрема, нейромережі сегментації [4] або Large Language Models – LLM[1].

Обчислювальні особливості нейромереж типу LLM

Останнім часом нейромережі класу LLM асоціюються із застосунками типу ChatGPT, які викликають широкі дискусії щодо методів використання як серед професіоналів, так і серед пересічних користувачів. Фактично застосунок складається із семантичної мовної моделі і машини виводу, яка генерує (добудовує, розширює або скорочує) тексти за певним запитом.

Машина виводу може добудовувати різну семантику з тих зв'язків, які означені у мовній моделі. Зв'язки у LLM побудовані за корпусом текстів, який покриває певну область людської діяльності. Для великих моделей, що базуються на великих корпусах текстів, або на великому обсязі текстів програмного забезпечення, побудова відповідей на запити нагадує роботу уявного штучного інтелекту, хоча результат такої роботи й не генерує нових знань (без специфічного програмування з боку користувача), а є лише редукцію або розширенням тексту за допомогою семантичних зв'язків.

У разі звуження корпусу текстів, над якими визначається модель для LLM, семантична модель фактично стає довідником з певної царини знань, і для застосунків у невеликій предметній області може буде використана для побудови реакцій [5] на стан зовнішнього середовища. LLM може розглядатися як аналог старого поняття «експертної системи», яка була більш формалізована, але давала лише ті відповіді, на які була запрограмована. LLM дозволяє мати більше свободи у реагуванні на ситуації реального життя.

Як приклад LLM можна навести NanoGPT[6], для якого є декілька мовних моделей, пояснення та приклади генерації мовних моделей за наборами текстів, а також вихідні коди машини виводу текстів, обсяг якої складає близько 800 рядків мовою C. Це дозволяє зацікавленим особам ознайомитися на тільки з алгоритмікою LLM, а й змоделювати швидкодію LLM.

Обчислювальний приклад

Як обчислювальний приклад розглянемо GPT2, описаний Basil Hosmer в [7], і більш детально в [8]. Використано GPT2 small configuration з проєкту Андрія Карпати NanoGPT [6] з параметрами: *layers* = 12, *heads* = 12, *embed* = 768. Параметр *embed* далі змінюватиметься для аналізу відповідно до розширених конфігурацій GPT2 у рамках здійснення обчислювальних експериментів.

У LLM цього типу основне обчислювальне навантаження відбувається в Attention Layer, найбільш інтенсивно використовується обчислення *attention*: наступна послідовність шести матричних множень [7,9], де знаком `@`, як і в оригінальному тексті, позначено матричне множення:

Таблиця 1.

```
(1)  Q = input @ wQ
(2)  K_t = wK_t @ input_t
(3)  V = input @ wV
(4)  attn = sdpa(Q @ K_t)
(5)  head_out = attn @ V
(6)  out = head_out @ wO
```

У наступній таблиці подано розмірності оброблюваних масивів та ємність пам'яті необхідна для їхнього зберігання. Базовий тип даних – 4-х байтове значення з плаваючою комою, стандартне для відеокарти. Конфігурація GPT2 – «*small*». Всі масиви розташовані у пам'яті відеокарти.

Масив	Стовп.	Рядків	Пам'ять, Кбт
Input_t	768	256	768
Input	256	768	768
wQ	768	64	192
Q	256	64	64
wK_t	64	768	192
wV	768	64	192
wO	64	786	192

Додатково використано допоміжні масиви у пам'яті відеокарти:

Масив	Стовп.	Рядків	Пам'ять, Кбт
K_t	64	256	64
V	256	64	64
Attn	256	256	256
head_out	256	64	64
Out	256	768	768

Сумарна пам'ять масивів моделі *small* складає 3.5Мбт, що, зважаючи на розмір кешів сучасних мобільних відеокарт, є досить великим об'ємом для кеш-пам'яті, але досить невеликим відносно всієї оперативної пам'яті відеокарти. У цьому випадку складність обчислювальної задачі залежить не від відносно маленького наразі об'єму використаної пам'яті, а від кількості доступів до пам'яті, оскільки кожна операція множення вимагає два доступи до

кеш-пам'яті, незалежно від способу оптимізації алгоритму. Всі способи програмної оптимізації намагаються зробити ефективнішим використання саме кеш-пам'яті, адже кошт доступу до кеш-пам'яті вважається мінімальним і в більшості випадків оптимізації не піддається. Але пропускна здатність кеш-пам'яті теж лімітована. Зважаючи на масований паралелізм, коли сотні потоків виконання можуть дати несподівані ефекти для пропускної здатності контролерів пам'яті від кеш першого рівня до динамічної пам'яті, нам необхідно використовувати спеціальне програмне забезпечення, наприклад, GPGPUSim [10].

Симуляція масованого паралелізму за допомогою GPGPUSim

На відміну від давньої звичної парадигми багатоядерного процесора (наприклад Intel Core, AMD Ryzen), відеокарта пропонує масований паралелізм (тисячі потоків виконання замість десятків у багатоядерному процесорі) за рахунок спрощення обчислювальних ядер, ускладнення ієрархії пам'яті та адаптованої концепції програмування. Зазначимо, що у випадку програмування відеокарти немає ускладнень технології паралельного програмування відносно програмування багатоядерного процесора, оскільки для отримання якісного паралельного коду необхідно знати тонкощі синхронізації пам'яті та роботи кеш-пам'яті для обох типів архітектур. GPGPUSim повністю і точно (максимум 5% відхилення по продуктивності від апаратної реалізації [10]) моделює роботу відеокарт Nvidia, що дозволяє досить точно оцінити продуктивність алгоритму, вплив оптимізацій на продуктивність та вплив апаратних особливостей на роботу алгоритму. Модель GPGPUSim гнучко налаштовується від параметрів конвеєризації арифметико-логічних пристроїв до часових діаграм доступу до мікросхем динамічної пам'яті. Так користувач може змінити необхідним для себе чином конфігурацію відеокарти і змоделювати виконання програми на будь-якій модифікації відеокарти.

Основною цікавинкою GPGPUSim є моделювання складної ієрархії пам'яті, оскільки повністю модельована часова діаграма доступу до всіх рівнів кеш і динамічної пам'яті, всі пристрої та черги синхронізації при доступі до кеш-пам'яті. Для нашого дослідження основними цікавими цифрами є обсяг доступу до кеш другого рівня (кеш L2), оскільки він практично найбільше впливає на швидкодію. Кеш першого рівня (L1), кеш третього рівня (L3) та доступ до основної пам'яті мають дещо менше, оскільки кеш L1 більш локально обслуговує шейдери, кеш L3 повільніший, а поведінка L2 дає більш релевантну інформацію.

GPGPUSim є прозорим для користувача, вбудовується в систему на базі Linux стандартним чином і використовується для запуску вже наявних програм для відеокарт, реалізованих за допомогою технології CUDA від Nvidia. Для цього GPGPUSim має власну версію бібліотеки *cudart*, яка підміняє собою як API CUDA, так і внутрішні механізми роботи драйвера за допомогою стандартних можливостей наявного завантажувача виконаних файлів формату ELF, які використовуються в усіх модифікаціях Linux. Симулятор формує шейдери, моделює стан пам'яті, API CUDA і OpenCL. Тому з точки зору користувача програма виконується на відеокарті, але вкрай повільно. Інформація про симуляцію перенаправляється до текстового файлу.

В нашому випадку використана Ubuntu 18, компілятор GCC версії 7.5.x для збірки симулятора і модельної програми, пакет утиліт (*nvcc* та інше) від CUDA 10.1. Була використана модель відеокарти QV100, яка має пасивне охолодження, тому за сценаріями використання більш схожа на мобільні відеокарти, і нема ніяких підстав вважати, що за кілька років обчислювальна потужність мобільних відеокарт не наблизиться до цієї моделі.

Для реалізації алгоритму, вказаного у табл. 1, була реалізована CUDA-програма, яка базована на прикладах множення матриць *matMul* з прикладів (*samples*), наданих до CUDA 10.1. Система збірки прикладу була модифікована для GPGPUSim шляхом додавання вказівок

стандартному лінкеру для використання бібліотеки *cudart* з комплекту GPGPUSim. Далі програма запускалася за допомогою GPGPUSim, дані про роботу кеш виділялися з файлу результатів симуляції.

Результати симуляції

Основною метою симуляції було дослідження алгоритму, представленого у табл. 1, спочатку на даних моделі GPT2 *small* і подальшого збільшення параметрів *layers*/*heads*/*embed* до моделей *large*/*xl*. Хоча алгоритм множення матриць є найбільш вивченим за всю історію досліджень, комплексні операції з матрицями на платформі з масованим паралелізмом з урахуванням складного кеша 2-го рівня цікаві з точки зору як ефективності роботи пам'яті, оскільки правильно використані кеші можуть працювати як на паралельну видачу даних, так і тих явищ у роботі кешу, які виникають у разі збільшення розміру даних і конфліктів у кеш, і, зважаючи на складність алгоритму, більш-менш точне їх обчислення неможливе без моделювання.

У наступних таблицях наведені результати для різних значень *embed*: 768, 1280 і 1600, використаних у різних конфігураціях GPT2 [6]. Стовпчик # означає номер множення у табл. 1, *L2 access* – кількість доступів до кеш пам'яті L2, *L2 misses* – кількість промахів у L2 (що додає сильну затримку у подачі даних), *L2 misses* % - відносна кількість промахів. Літера *M* означає мільйон.

Таблиця 2

Доступ у пам'ять для *embed* = 768

#	L2 access	L2 misses	L2 misses %
1	1.23M	0.053M	4.3%
2	1.59M	0.098M	6.2%
3	2.82M	0.108M	3.8%
4	2.83M	0.108M	3.8%
5	2.97M	0.108M	3.8%
6	2.98M	0.108M	3.6%

Для моделі GPT2 *small* ми бачимо, що в цілому відносна кількість промахів у L2 є великою лише для стадії 2 алгоритму, оскільки для великої кількості алгоритмів «прийнятні» цифри відносних непопадань у L2 це 2-3%. Більший відсоток призводить до деградації швидкодії відносно наявних можливостей арифметико-логічних пристроїв комп'ютера. Аналізуємо далі.

Таблиця 3

Доступ у пам'ять для *embed* = 1280

#	L2 access	L2 misses	L2 misses %
1	3.22M	0.190M	5.9%
2	4.11M	0.305M	7.4%
3	7.46M	0.592M	7.9%
4	7.47M	0.597M	8.0%
5	7.61M	0.615M	8.1%
6	7.62M	0.615M	8.1%

Набір даних збільшився у 1.67 рази, відсоток промахів у кеш значно збільшився, особливо на останніх стадіях (4-6). Для стадій 2-6 спостерігаємо дуже великий відсоток промахів, понад 7, що призводить до значних сповільнень алгоритму, адже кошт промаху у десятки разів перевищує кошт вибірки даних з кешу L2. Проаналізуємо велику модель *xl* з GPT2.

Таблиця 4

Доступ у пам'ять для *embed*=1600

#	L2 access	L2 misses	L2 misses %
1	5.42M	0.315M	5.8%
2	6.96M	0.655M	9.4%
3	12.4M	1.08M	8.7%
4	12.5M	1.08M	8.7%
5	12.5M	1.1M	8.8%
6	12.5M	1.1M	8.8%

Порівняно із табл. 3 відсоток промахів збільшився, але принципово дані табл. 3 і 4 вказують на те, ще така конфігурація алгоритму перевищує продуктивність такої конфігурації процесора.

Наведемо також розбивку доступу до пам'яті на читання і запис для *embed*=768 і 1280.

Таблиця 5

#	*embed*=768		*embed*=1600	
	L2 RD	L2 WR	L2 RD	L2 WR
1	1.17M	0.074M	5.1M	0.32M
2	1.44M	0.147M	6.4M	0.64M
3	2.6M	0.221M	11.5M	0.96M
4	2.61M	0.222M	11.4M	0.96M
5	2.74M	0.23M	11.5M	0.96M
6	2.75M	0.23M	11.7M	0.97M

Ці цифри типові для матричних операцій, коли на велику кількість операцій читання маємо небагато операцій запису.

Дані табл. 2, 3 і 4 фактично відрізняються лише однією розмірністю масивів *embed*, використаних в алгоритмі в табл. 1. Всього цей алгоритм використовує три розмірності, і було вирішено як обчислювальний експеримент збільшити третю з матричних розмірностей з 64 до 96.

Таблиця 6

Доступ у пам'ять для *embed*=1600 і розмірності рядків Q і 96

#	L2 access	L2 misses	L2 misses %
1	5.42M	0.297M	5.5%
2	7.59M	0.646M	8.5%
3	13.01M	1.07M	8.2%
4	13.03M	1.08M	8.3%
5	13.17M	1.09M	8.3%
6	13.19M	1.09M	8.3%

Порівнюючи табл. 4 і 6 зазначимо, що, незважаючи на більший обсяг пам'яті, з яким оперує алгоритм у конфігурації з табл. 6, відносна кількість промахів у L2 кеш у відсотках стає меншою. Оскільки алгоритм досить складний, немає сенсу робити теоретичні розвідки щодо поведінки кешу, але симулятор дозволяє виявляти такі цікаві ефекти. Важливо, що симулятор моделює поведінку контролеру пам'яті між запусками шейдерів і таким чином може вказувати на оптимальні сценарії запуску шейдерів, коли виконані шейдери ефективно використовують результати попередньо виконаних шейдерів.

Зібрана у табл. 2-6 інформація може піддаватися вільній інтерпретації, але підкріпимо дані симуляції кешу L2 даними щодо кількості доступів до динамічної пам'яті. Трафік у динамічну пам'ять (DRAM) скерований контролером кеш пам'яті і є необхідним для збереження даних, які не вміщуються у кеш. Зазначимо, що GPGPUSim симулює DRAM за допомогою повного точного опису доступу до пам'яті у прив'язці до реального часу. Таким чином, трафік у DRAM корелює з кількістю промахів у L2 кеш, але не у лінійній залежності.

Таблиця 7.

Кількість читань та записів DRAM

	embed = 1280		*embed* = 1600	
	Reads	Writes	Reads	Writes
1	17.8K	42.7K	38.5K	156K
2	29K	124K	55K	462K
3	111K	383K	157K	834K
k	115K	387K	162K	834K
5	131K	403K	177K	853K
6	131K	403K	177K	853K

Окремо зазначимо, що трафік у DRAM для *embed*=768 нульовий. Це пояснюється тим, що у разі відсутності запитів на виділення кешу (які генеруються під час роботи інших шейдерів, які виконують свої певні алгоритми) контролер кеш не робитиме запитів на збереження даних у динамічній пам'яті окрім специфічних синхронізацій. Коли використаний обсяг пам'яті не вміщується в кеш, з'являється трафік у DRAM. Оскільки алгоритм з табл. 1 виконується багато раз, трафік у пам'ять відповідно зростає.

Наступний рис. 1 ілюструє залежність промахів у кеш-пам'ять від параметра *embed*, із зростанням якого зростає розмір масивів.

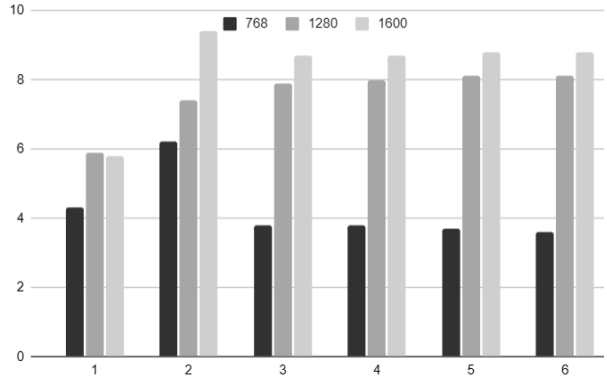

Рис. 1. Залежність промахів у кеш пам'ять від параметра *embed*.

Якість роботи алгоритму GPT2 прямо залежить від величини параметру *embed*. Якщо ми звернемося до більш потужних моделей, що обробляють дуже великі корпуси текстів[11] (наприклад, частину мережі Internet), то розмір наборів даних вимагає для тренування гігабайти пам'яті, а для виводу – 16Гбайт оператив-

ної пам'яті. Проте метою більш великих моделей є наближення до людського стилю викладу інформації, а GPT2 має меншу ресурсоємність і може продукувати вивід на відносно малопотужних процесорах. Відносно невелика швидкість генерації відповідей (десятки токенів на секунду) на сучасному процесорі Intel Core фактично вимагає переміщувати код на відеопроцесор і підлаштовувати розмір моделі до об'єму наявної пам'яті.

Розвитком цієї роботи можуть кілька напрямів. По-перше, визначення необхідних обчислювальних потужностей, необхідних для запуску алгоритму в цілому, оскільки темою цієї статті були найбільш обчислювально важкі місця. Це дозволяє визначити типовий час відповіді у режимі реального часу. По-друге, визначення прийнятного розміру корпусу професійних (спеціалізованих), за допомогою якого GPT2 дозволяє генерувати або звужувати текст, дозволяє визначити ліміти застосовності алгоритмів такого типу на мобільних пристроях. Таке дослідження стикається зі складнощами, оскільки моделі на основі вірогідностей важко тестувати, складно визначити якість проробки моделлю запиту, необхідно визначити обмеження на запити. Але це окреме дослідження ресурсоємності алгоритмів LLM показує, що ці алгоритми можуть використовуватися не лише на великих і потужних обчислювальних системах.

Висновки

У статті розглянуто аналіз продуктивності нейромереж типу LLM для сучасної відеокарти. Показано, що невеликі конфігурації моделі GPT2 можуть бути розміщені на обчислювальних пристроях такого типу, що відкриває можливості використання цього типу нейромереж для автономної роботи на мобільних пристроях.

References

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L .Jones, A. Gomez, L. Kaizer, I. Polosukhin. Attention is all you need. // In proc. 31st Conf. on Neural Information Processing Systems (NIPS), Dec. 2017, pp. 6000-6010.
2. S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro B, et al. CuDnn: efficient primitives for deep learning. arXiv preprint: arXiv:14100759. (2014). [Accessed 17/03/2024]
3. Rahozin D., Doroshenko A. (2022) Performance Model for Convolutional Neural Networks. In: Shkarlet S. et al. (eds) Mathematical Modeling and Simulation of Systems. MODS Lecture Notes in Networks and Systems, vol 344. Springer, pp. 239-251. DOI: doi.org/10.1007/978-3-030-89902-8_19
4. Y. Guo, Y. Liu, T. Georgiou et al. A review of semantic segmentation using deep neural networks. Int J Multimed Info Retr 7, 87–93 (2018). doi.org/10.1007/s13735-017-0141-z
5. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan and Y Cao. ReAct: Synergizing Reasoning and Acting in Language Models. (2022) arxiv.org/abs/2210.03629. [Accessed 17/03/2024]
6. A. Karpathy A. NanoGPT. Available from: github.com/karpathy/nanoGPT [Accessed 13/04/2024]
7. B. Hosmer. Inside the Matrix: Visualizing Matrix Multiplication, Attention and Beyond. (25 Sept 2023) Available from: pytorch.org/blog/ inside-the-matrix/ [Accessed 13/03/2024]
8. A. Golden, S. Hsia, F. Sun, B. Acun, B. Hosmer, Y. Leeet al. Generative AI Beyond LLMs: System Implications of Multi-Modal Generation. (2023) Av ailable from: arxiv.org/abs/2312.14385 [Accessed 13/04/2024]
9. M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, B. Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. (2020) Available at: arxiv.org/abs/1909.08053 [Accessed 13/04/2024]
10. J. Lew, D. Shah, S. Pati, S. Cattell, M. Zhang et all. Analyzing Machine Learning Workloads Using a Detailed GPU Simulator (ISPASS 2019) Available at: arxiv.org/abs/1811.08933 [Accessed 13/04/2024]
11. T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal et all. Language Models are Few-Shot Learners. // In Proc. of 34th Conf. on Neural Information Processing Systems (Dec 2020), Vancouver, Canada. P. 1877-1901. Doi: 10.5555/3495724.3495883

Одержано: 05.04.2024
Внутрішня рецензія отримана: 14.04.2024
Зовнішня рецензія отримана: 20.04.2024

Про авторів:

Рагозін Дмитро Васильович,
кандидат технічних наук,
старший науковий співробітник.
http://orcid.org/0000-0002-8891-7002.

Дорошенко Анатолій Юхимович, доктор фізико-математичних наук, професор,
завідувач відділу теорії комп'ютерних обчислень
http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
проспект Перемоги 37
E-mail: Dmytro.Rahozin@gmail.com